HIGHLY PIPELINED PRINTING SYSTEM ARCHITECTURE

Technical Field of the Invention

The present invention relates generally to printing systems, and in particular, to high performance printing systems. The present invention relates to a method and apparatus for pipeline data processing in printer systems.

Background Art

Conventional desktop and mainframe computer-based printing systems typically operate by processing pages in a document in a sequential manner. Accordingly, each page is individually processed, the individual processing for a particular page having to be completed before processing for a following page commences. As a consequence, the processing for a page does not begin until the previous page has been completely printed. The printing process itself typically consists of a number of sequential processing stages, each of which is made up of a combination of computer software and hardware based sub-processes. In the conventional printing model, each sub-process stage must be fully completed before the next sub-process stage can commence.

Many modern printer engines are capable of printing multiple pages during a single printing cycle. These engines must, however, be fed data continuously, in order to maintain the continuous printing cycle, and thus perform at their maximum rated engine speed and throughput. If a situation arises where raw data required for printing a particular page is not ready when required, the printer engine must cycle down to a standby mode in an orderly fashion, and cease feeding input media, such as paper, in order to prevent blank media appearing in the middle of the output document. Only when raw data is again available for the subsequent page, can the printer engine again begin processing.

This cycle down/cycle up process reduces engine throughput considerably, and of course, causes the printing system to operate below maximum performance levels. The presence of bottlenecks which are inherent in conventional sequential printing systems makes it difficult to provide a continuous supply of raw print data to the printer engine as required. As printer engine speeds increase, this problem is exacerbated, since the speed

with which the raw data must be supplied to the print engine also increases. Furthermore, as achievable print resolution increases, the volume of data processed in the course of each print job also increases, typically increasing in proportion to the square of the resolution increase, since the printed page is two-dimensional. The aforementioned technological advances place increasing strain on the conventional sequential processing model.

One present solution to the aforementioned problems is to "spool" job data at various stages in the print data flow processing. This approach can sometimes eliminate the cycle down/cycle up problem if applied to a relatively short print run. A penalty is incurred, however, since spooling adds additional overhead to the print data processing, because data must be typically copied to, and from, a relatively slow storage device such as a hard disc. Furthermore, spooling introduces an arbitrary restriction on the length of a print job, since data for an entire job must typically be stored in spooling systems. Furthermore, the spooling approach also introduces additional start-up delay, before the first page of a job can be printed.

Summary of the Invention

It is an object of the present invention to substantially overcome, or at least ameliorate, one or more disadvantages of existing arrangements.

According to a first aspect of the invention, there is provided a printing system for printing a print job comprising a number of pages, the system including:

a receiving hardware interface for performing first coupling of incoming page description data, in the form of a display list, to a renderer software interface ;

said renderer software interface for performing first processing of the display list to thereby output a first processed display list; and

rendering hardware, adapted to perform second processing of the first processed display list to thereby output raw pixel data, wherein:

said receiving hardware interface, the rendering software interface, and the renderer hardware are arranged to operate in a pipelined manner, being thereby capable of concurrently processing job data from at least one page of the print job.

According to another aspect of the invention, there is provided a method of data processing for a printing system adapted for printing a print job comprising a number of pages, said method comprising steps of:

first coupling, by a receiving hardware interface, of incoming page description data to a renderer software interface in the form of a display list;

performing first processing, by the renderer software interface, of the display list to thereby output a first processed display list: and

performing second processing, by rendering hardware, of the first processed display list to thereby output raw pixel data, wherein:

said first coupling, said first and said second processing steps operate in a pipelined manner, being thereby capable of concurrently processing job data from at least one page of the print job.

According to another aspect of the invention, there is provided a computer readable medium for storing a program for a print system adapted to print a job comprising a number of pages, said program comprising:

code for a first coupling step for coupling, by a receiving hardware interface, of incoming page description data to a renderer software interface in the form of a display list;

code for a first processing step for processing, by the renderer software interface, of the display list to thereby output a first processed display list; and

code for a second processing step for processing, by rendering hardware, of the first processed display list to thereby output raw pixel data, wherein:

said code for the first coupling step, said code for the first and said second processing steps operate in a pipelined manner, being thereby capable of concurrently processing job data from at least one page of the print job.

According to another aspect of the invention, there is provided a computer program for a print system adapted to print a job comprising a number of pages, said program comprising:

code for a first coupling step for coupling, by a receiving hardware interface, of incoming page description data to a renderer software interface in the form of a display list;

code for a first processing step for processing, by the renderer software interface, of the display list to thereby output a first processed display list; and

code for a second processing step for processing, by rendering hardware, of the first processed display list to thereby output raw pixel data, wherein:

said code for the first coupling step, said code for the first and said second processing steps operate in a pipelined manner, being thereby capable of concurrently processing job data from at least one page of the print job.

According to another aspect of the invention, there is provided a method of data processing for a printing system which comprises a sequence of pipeline processes, said method comprising, for a current pipeline process, steps of:

reading input data from an upstream pipeline process;

operating upon said input data if an internal buffer of said current pipeline process is not full;

stalling said upstream pipeline process, if said internal buffer is full; and

writing said input data, having operated thereupon, to a downstream pipeline process, if said downstream pipeline process is not stalling said current pipeline process.

Brief Description of the Drawings

One or more embodiments of the present invention will now be described with reference to the drawings, in which:

Fig. 1 shows a block diagram representation of hardware components in a printing system according to a first arrangement;

Fig. 2 shows a pipeline arrangement for the host computer of Fig. 1;

Fig. 3 shows pipeline processing details for the printer controller of Fig. 1;

Fig. 4 shows pipeline details for the print engine of Fig. 1;

Fig. 5 is a schematic block diagram of a general purpose computer upon which the disclosed arrangement can be practised;

Fig. 6 is a flowchart of method steps relating to buffer-based processing of data from an upstream device;

Fig. 7 is a flowchart of method steps relating to buffer-based processing of data for a downstream device;

Fig. 8 is a flowchart of method steps relating to generic stall-based processing of data from an upstream device; and

Fig.9 is a flowchart of method steps relating to generic stall-based processing of date for a downstream device.

Detailed Description including Best Mode

Where reference is made in any one or more of the accompanying drawings to steps and/or features, which have the same reference numerals, those steps and/or features have for the purposes of this description, the same function(s) or operation(s), unless the contrary intention appears.

In the context of this specification, the term "line" is used to denote a data connection between process blocks, and/or between sub-process blocks. Data connections can take a variety of different, but functionally equivalent physical forms, including hard-wired and wired connections, and can also represent software calls between software processes.

Fig. 1 shows major hardware components of a printing system, according to a first arrangement. The system supports both a forward data flow depicted by an arrow 110, and a reverse data flow depicted by an arrow 112. A host computer 100 incorporates a host interface 102, which provides an operative coupling between the host computer 100 and a printer controller 104. The printer controller 104, in turn, incorporates an engine interface 106 which provides an operative coupling to a print engine 108. The print engine 108 is operable for the feeding of output media, and marking print data thereon, to thus perform a printing function, the output media typically being paper. Typically, the printer controller 104 and the print engine 108 are regarded as a single entity by an end user, however such can often be sourced from different Original Equipment Manufacturers (OEMs). Figs. 2 – 4 will first be described in terms of

processes acting on the forward data path 110, and thereafter the reverse data path 112 will be described. The various components shown in Fig. 1 may be performed by a combination of hardware and software, the extent of combination typically being dependent upon the desired functions of the specific component. Typically, a physical combination of the printer controller 104 and the print engine 108 is called a "printer".

Fig. 2 illustrates pipeline processes operating within the host computer 100 as described in the above arrangement, and in particular, relates to processes on the forward data path 110. Fig. 2 elaborates on the various data processing stages within major hardware and software components depicted in Fig. 1. The processing stages depicted in Fig. 2 are configured to operate sequentially but concurrently, in which a current stage can accept data from a previous stage, and send data to a downstream stage as output data from the current stage becomes available. Each stage also provides a "stall control" signal back to the previous (upstream) stage, in order to indicate when data flow should be quenched to avoid causing overflow of internal buffers in the downstream stage or stages of the pipeline.

Accordingly, having regard to a page to be printed, a host application 200 makes printer Graphical Device Interface (GDI) calls on a line 202, these calls describing the page image to a printer driver 204. GDI calls are used in Windows™ operating systems manufactured by Microsoft Corporation of USA. The host application 200 uses a GDI call to tell the printer driver 204 about the appearance of the page to be printed. GDI calls can, for example, direct the printer driver 204 to draw a circle, line, picture or other graphical object at a specific location on the output page. The printer driver 204 in turn generates a page description for communication on a line 206, this description being produced in any applicable Page Description Language (PDL) accepted by the printer. Typical page description languages include Postscript, PCL, LIPS, or alternatively, printer specific formats which can be proprietary. The page description generated by the printer driver 204 is written to a spooler 208, which may be part of a Windows™ printing system manufactured by Microsoft Corp USA. The spooler 208 in the present arrangement sends the data, using a line 210, directly to a printer port monitor 212,

without a need to spool the data onto a disk drive memory of the computer 100. The port monitor 212 in turn sends the data, using a line 214, to an applicable host hardware interface 216, performing the functions of the interface 102, and which can be any interface capable of communication with the printer. Examples of such interfaces include IEEE-1284, TCP/IP over Ethernet, or any other suitable printer interface. Job data from the hardware interface 216 is sent on a line 218 to a printer controller receiving hardware interface 300 (see Fig. 3).

Fig. 3 shows pipeline processing details for the printer controller 104 of Fig. 1 A receiving hardware interface 300 receives job data on the line 218 and sends the job data on a line 302 to a page description language interpreter 304, which interprets the contents of the job data on arrival, in order to generate and send a display list. In one arrangement, the interpreter 304 generates a display list whose content is substantially identical to the job data received on the line 302, with merely the header and other overhead information removed. In another arrangement, the interpreter 304 interprets the job data on the line 302 to produce a display list which is substantially different to the received PDL job data. The display list is transferred using a line 306 to a renderer software interface 308. The display list which is received by the renderer software interface 308 on the line 306 is processed into a form suitable for rendering and sent, on a line 310, to a rendering hardware process 312. The renderer software interface 308 can take a number of different forms depending upon the arrangement of the description language interpreter 304. If the interpreter 304 performs, as described, a header-stripping operation only, then the renderer software interface 308 merely controls the rendering hardware process 312, performing little, if any additional processing. If on the other hand, the interpreter 304 performs a more comprehensive interpretation function, then the renderer software interface 308 performs a correspondingly more complex processing to bring the display list into a form suitable for rendering. The rendering hardware process 312 stores and retrieves the data received on the line 310 in a display list memory 324. The storage and retrieval in the display list memory 324, which is depicted by a bilateral arrow 328, is performed using, possibly, Direct Memory Access (DMA), and the

rendering hardware processes 312 generates raw pixel data for transmission from the printer controller 104 using a line 314. The raw pixel data which is sent on the line 314 can be represented, for example, in the RGB colour space, and is sent to a colour converter 400 of the print engine 108, as seen in Fig. 4.

Fig. 4 shows pipeline process details for the print engine 108. The incoming raw pixel data which is received on the line 314 is input into a colour converter 400, which converts the data to a colour space as required by a subsequent marking process 408, such as CMYK. The colour converter 400 outputs the CMYK (or alternative) pixel data on a line 402 to an output processing sub-process 404, which generates host processed marker data for sending on a line 406 as required by the marking engine 408. The marking engine 408 generates a final image on the appropriate output medium.

The above description has been directed to the forward data flow path 110 shown in Fig. 1 from the host application 200 and through to the marking engine 408. It is noted that each stage in the aforementioned cascade of sub-processes typically has some internal buffering. Each stage can, in addition, signal a "stall condition" back to an upstream stage in the pipeline, should the corresponding internal buffer become full. This signalling takes place in the direction of the reverse data flow path 112 shown in Fig. 1 and makes use of the lines 220-228 of Fig. 2, lines 316-322 of Fig. 3 and lines 410 and 412 of Fig. 4.

It is noted that the afore-described processing stages can be implemented using a mix of hardware and/or software. The "signalling" between stages will, accordingly, be either a hardware signal on a connecting line (for example between two hardware stages), or alternatively the "signal" takes the form of a software call, or indication, (eg. between two software stages). This applies both to signals on the forward path 110, and on the reverse path 112 which relates to signalling of the stall conditions.

Stall signals flow in the reverse direction 112, and are depicted as dashed arrows in the figures, indicating that these signals are only generated, in the present arrangement. if they are necessary to prevent buffer overflow. Other stall mechanisms are described in relation to Fig. 8. The forward data flow 110 is depicted by forward arrows having solid

lines, indicating that this data is expected to flow in the normal course of printer operation. Stall condition signalling prevents data overflow from occurring in any processing stage, while at the same time allowing maximum concurrent processing to proceed.

The stall control mechanism, considered from the marking engine 408 end of the series of cascaded sub-processes in Fig. 4, operates as follows. The marking engine 408 accepts data synchronously from the output processing stage 404 on the line 406 as has been described. When the marking engine 408 is busy, it signals this "busy" state via a hardware signal depicted by a dashed arrow 410 to the output processing stage 404 which in turn stalls the colour converter process 400 via a hardware signal on a dashed arrow 412.

The colour converter 400 consequently stalls the rendering hardware sub-process 312 (see Fig. 3) by means of a signal on a dashed arrow 316. The rendering hardware 312, consequently stalls the renderer software interface 308, by means of a signal on a dashed arrow 318, this being performed if the display list memory 324 becomes full. This stall signal on the line 318 propagates further in the reverse direction to the interpreter sub-process 304 on a dashed arrow 320, blocking further write operations to the renderer 308. In turn, the interpreter 304 ceases reading input date from the receiving hardware interface 300, this being caused by a stall signal on a dashed arrow 322. The receiving hardware interface 300, in turn, stalls the host hardware interface 216 (see Fig. 2) by means of a signal on a dashed arrow 220, causing the host hardware interface 216 to block write calls from the port monitor 212 by means of a stall signal on the a dashed arrow 222. The port monitor 212, in turn, blocks write calls from the spooler 208 by means of a stall signal on a dashed arrow 224, this in turn being propagated to block write signals from the printer driver 204 by means of a stall signal on a dashed arrow 226. The printer driver 204, in turn, prevents the GDI calls from the host application 200 by means of a stall signal on a dashed arrow 228.

It should be noted that each of the aforementioned stall signals is asserted when a relevant downstream module cannot accept any further data, typically because an internal

buffer associated with that downstream module is full. In this manner, for example, a stall signal generated by the marking engine 408 (see Fig. 4) can propagate all the way back to the host application 200 (see Fig. 2). It should, however, be noted that stall signals can originate anywhere along the series of pipelined sub-processes, and indeed can be generated at more than one location substantially simultaneously.

The printing system arrangement described in relation to Figs. 2 to 4 operates in an end-to-end pipelined manner. Accordingly, the subprocesses 200, 204, 208, 212, 216, 300, 304, 308, 324, 313, 400, 404 and 408 are capable of concurrently processing job data from at least one page of the print job. This end-to-end concurrent processing on the forward path 110 (see Fig. 1), which is facilitated by the propagation of stall control signals on the reverse path 112, provides a significant advantage in the throughput of the described printing arrangement.

The disclosed method of data processing for printer systems can be implemented in dedicated hardware elements, or in dedicated software elements, or using a mix of hardware and software elements for the aforementioned process stages. Having regard to the particular implementation depicted in Figs. 2 to 4, processing stages 200, 204, 208 and 212 are typically implemented in software. Accordingly, the interposed communication "connections", eg. 202 and 228, take the form of software calls, or indications. The interface processing stage 216 is typically implemented in a mixture of software and hardware. Accordingly, the connections 218 and 220 can take the form of a hardware/software mix, depending on the specifics of the interfaces used (eg. IEEE 1284 Ethernet, and so on). Processes 304 and 308 are typically implemented in software, while processes 324 and 312 are typically implemented in hardware. The connections between these various stages carry either hardware based signals, or software calls, in accordance with the specifics of the aforementioned implementation.

The method of data processing for a printing system may be implemented in dedicated hardware such as one or more integrated circuits performing the functions or sub functions of data processing for a printing system. Such dedicated hardware may

include graphic processors, digital signal processors, or one or more microprocessors and associated memories.

The method of data processing for a printing system can also be practiced using a conventional general-purpose computer system 500, such as that shown in Fig. 5 wherein the processes of Figs. 6 and 7 may be implemented as software, such as an application program executing within the computer system 500. In particular, the steps of data processing for a printing system are effected by instructions in the software that are carried out by the computer. The software may be divided into two separate parts, one part for carrying out the data processing for printing system methods, and another part to manage the user interface between the latter and the user. The software may be stored in a computer readable medium, including the storage devices described below, for example. The software is loaded into the computer from the computer readable medium, and then executed by the computer. A computer readable medium having such software or computer program recorded on it is a computer program product. The use of the computer program product in the computer effects an apparatus for data processing for a printing system in accordance with the described arrangements.

The computer system 500 comprises a computer module 501, input devices such as a keyboard 502 and mouse 503, output devices including a printer 515 and a display device 514. A Modulator-Demodulator (Modem) transceiver device 516 is used by the computer module 501 for communicating to and from a communications network 520, for example connectable via a telephone line 521 or other functional medium. The modem 516 can be used to obtain access to the Internet, and other network systems, such as a Local Area Network (LAN) or a Wide Area Network (WAN).

The computer module 501 typically includes at least one processor unit 505, a memory unit 506, for example formed from semiconductor random access memory (RAM) and read only memory (ROM), input/output (I/O) interfaces including a video interface 507, and an I/O interface 513 for the keyboard 502 and mouse 503 and optionally a joystick (not illustrated), and the interface 508 for the modem 516. A storage device 509 is provided and typically includes a hard disk drive 510 and a floppy disk

drive 511. A magnetic tape drive (not illustrated) may also be used. A CD-ROM drive 512 is typically provided as a non-volatile source of data. The components 505 to 513 of the computer module 501, typically communicate via an interconnected bus 504 and in a manner which results in a conventional mode of operation of the computer system 500 known to those in the relevant art. Examples of computers on which the embodiments can be practised include IBM-PC's and compatibles, Sun Sparcstations or alike computer systems evolved therefrom.

Typically, the application program of the embodiment is resident on the hard disk drive 510 and read and controlled in its execution by the processor 505. Intermediate storage of the program and any data fetched from the network 520 may be accomplished using the semiconductor memory 506, possibly in concert with the hard disk drive 510. In some instances, the application program may be supplied to the user encoded on a CD-ROM or floppy disk and read via the corresponding drive 512 or 511, or alternatively may be read by the user from the network 520 via the modem device 516. Still further, the software can also be loaded into the computer system 500 from other computer readable medium including magnetic tape, a ROM or integrated circuit, a magneto-optical disk, a radio or infra-red transmission channel between the computer module 501 and another device, a computer readable card such as a PCMCIA card, and the Internet and Intranets including email transmissions and information recorded on websites and the like. The foregoing is merely exemplary of relevant computer readable mediums. Other computer readable mediums may be practiced without departing from the scope and spirit of the invention.

Fig. 6 shows a flowchart of method steps for a buffer-based port process 610 which "runs" on an upstream input port (such as the port 230, see Fig. 2) of a representative pipeline process (such as the printer driver 204 in this example) described in Figs. 2 to 4. On a point of terminology, the "port process" 610 is to be differentiated from the "pipeline process" 204 (namely the printer driver 204) on whose upstream input port 230 the port process 610 runs. In a step 600, the port process 610 waits for input data from an upstream pipeline process. Thereafter, in a testing step 602, an internal buffer of

the pipeline process 204 on whose upstream port the port process 610 is running is checked for overflow. If an overflow condition is detected, then the port process 610 is directed in accordance with a "yes" arrow to a stalling step 604, which stalls the upstream device (namely the host application 200) until buffer space in the pipeline process 204 becomes available. Once such space becomes available, the port process 610 is directed to a step 606 which proceeds to read input data from the upstream device into the internal buffer of the pipeline process 204. Returning to the testing step 602, if the internal buffer of the pipeline process 204 is not experiencing an overflow condition, then the port process 610 is directed in accordance with a "no" arrow to the step 606. After the step 606 reads input data from the upstream device into the internal buffer of the pipeline process 204, the port process 610 is directed in accordance with an arrow 608 back to the step 600, where the port process 610 waits for further input data from the upstream process.

Fig. 7 shows a flowchart of method steps describing a buffer-based port process 710 which runs at the output (namely downstream facing) port (such as the port 232, see Fig. 2) of each pipeline process (such as the host hardware interface pipeline process 216) described in Figs. 2 to 4. In a step 700, the port process 710 waits for data to appear in an internal buffer of the pipeline process 216. Once such data is detected, a testing step 702 considers whether the output of the pipeline process 216 is being stalled. If this is the case, then the port process 710 is directed in accordance with a "yes" arrow to a wait step 704, which freezes operation of the pipeline process 216 until the downstream device removes the present stall condition. Thereafter, the port process 710 is directed to a write step 706, which writes data from the internal buffer of the pipeline process 216 to the output port thereof. Returning to the testing step 702, if the output is not being stalled (ie. by the downstream device), then the process 710 is directed in accordance with a "no" arrow to the step 706. After step 706 the process 710 is directed in accordance with an arrow 708 back to the wait step 700.

Fig. 8 shows a flowchart of method steps for a generic stall-based port process 810 which "runs" on an upstream input port (such as the port 230 of Fig. 2) of a

representative pipeline process (namely the printer driver 204) described in Figs. 2 to 4 In a step 800, the port process 810 waits for input data from an upstream pipeline process. Thereafter, in a testing step 802, a test is performed for a "stall condition". Although Figs. 6 and 7 depicted specific buffer-based stall processes, in fact a number of different mechanisms can lead to a general stall condition. Thus, for example, if a current process is too busy to read from an upstream process, then a stall condition will exist. In this case, the current process is too busy to either read new data from an upstream process, or to write processed output data to a corresponding downstream process. Alternately, a local buffer in the current process can be full, thereby leading to a stall condition, as described in relation to Figs. 6 and 7. It is noted that in this case, the current process can write processed data out to a downstream process, however cannot read new input data from a corresponding upstream process. Alternately, a downstream process can itself be stalled, consequently propagating a stall condition back in the upstream direction to the current process. In this event, the current process is unable to write out to the stalled downstream process. Returning to Fig. 8, if a stall condition is detected, then the port process 810 is directed in accordance with a "yes" arrow to a stalling step 804, which stalls the upstream device (ie the host application 200) until the stall condition terminates. Once this occurs, the port process 810 is directed to a step 806 which proceeds to read and process input data from the upstream device. Returning to the testing step 802, if no stall condition exists, then the port process 810 is directed in accordance with a "no" arrow to the step 806. After the step 806 reads and processes input data from the upstream device, the port process 810 is directed in accordance with an arrow 808 back to the step 800, where the port process 810 waits for further input data from the upstream process.

Fig. 9 shows a flowchart of method steps describing a generic stall-based port process 910 which runs at the output (ie. downstream facing) port (such as the port 232, see Fig. 2) of each pipeline process (eg the host hardware interface pipeline process 216) described in Figs. 2 to 4. In a step 900, the port process 910 waits for processed data to be available as a result of processing by the current process. Once such data is detected, a

testing step 902 considers whether a stall condition exists. If this is the case, then the port process 910 is directed in accordance with a "yes" arrow to a wait step 904, which freezes operation of the pipeline process 216 until the stall condition terminates. Thereafter, the port process 910 is directed to a write step 906, which writes processed data to the corresponding downstream process. Returning to the testing step 902, if no stall condition is present then the process 910 is directed in accordance with a "no" arrow to the step 906. After step 906 the process 910 is directed in accordance with an arrow 908 back to the wait step 900.

In general, each of the pipeline processes shown in Figs. 2 to 4 (from the printer driver 204 to the output processing device 404) runs an associated input port process 810 and an associated output port process 910 on the corresponding input and output ports respectively of the pipelined process.

Industrial Applicability

It is apparent from the above that the embodiment(s) of the invention are applicable to the printing industry, and indeed to any industry in which high performance printing processes are used or required.

The foregoing describes only one embodiment of the present invention, and modifications and/or changes can be made thereto without departing from the scope and spirit of the invention, the embodiment being illustrative and not restrictive.